REDTAIL LABORATORY EXERCISE

Rain-Responsive Wiper Controller

STM32 NUCLEO-WB55RG with Mbed OS

LEVEL **Basic** | TIME **45-75 minutes** | PLATFORM **STM32 NUCLEO-WB55RG**

Implement and test a one-bit windshield-wiper controller on the STM32 NUCLEO-WB55RG. Your Mbed application decides whether the blade should move; the simulation owns its angle and automatic endpoint reversal.

The Wiper 3D simulation and original remote-laboratory exercise were created by Giovanna Lani and WebLab-Deusto at the University of Deusto in Spain. This lesson adapts their rain-driven challenge to the STM32/Mbed workflow and the current REDTAIL mapping. The completed application is available separately to verified instructors.

Figure 1. Wiper 3D simulation by Giovanna Lani and WebLab-Deusto, University of Deusto (Spain).

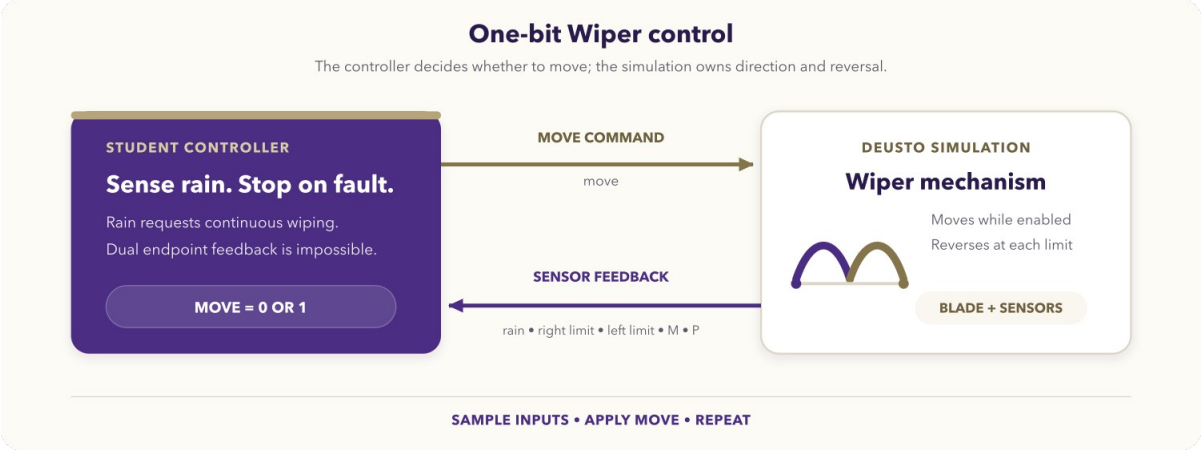

Figure 2. One-bit Wiper control loop.

Learning objectives

After completing the activity, you should be able to:

- configure Mbed digital inputs and outputs for a remote simulation;
- implement a deterministic Boolean control policy;
- use one input snapshot per polling-loop iteration;
- distinguish controller logic from simulation-owned motion;
- stop conservatively on contradictory endpoint feedback;
- test rain, pause, resume, automatic reversal, and fault behavior.

Prerequisites

You should know basic C++, Mbed `DigitalIn` and `DigitalOut`, and polling loops. You need an Mbed OS 6 or Mbed OS CE environment for `NUCLEO_WB55RG` and access to the Wiper simulation.

Open the REDTAIL Wiper simulation and the current STM32 mapping before starting.

System behavior

Input	Meaning in the current simulation
`rainSensor`	Rain is active and automatic wiping is requested.
`rightSensor`	The blade is within the right endpoint range.
`leftSensor`	The blade is within the left endpoint range.
`mButton`	Optional M button; not required by the core assignment.
`pButton`	Optional P button; not required by the core assignment.

Output	Meaning
`move`	High allows movement; low pauses the blade.

The core application must move during rain, stop while dry, stop when both endpoint inputs are high, leave normal direction/reversal to the simulation, and ignore M/P.

STM32 signal mapping

Directions are relative to the Mbed application.

Runtime signal	Mbed pin	Direction
Rain sensor	`PC_13`	Input
Right endpoint	`PA_6`	Input
Left endpoint	`PB_9`	Input
M button	`PB_8`	Input
P button	`PC_12`	Input
Move	`PC_4`	Output

Optional LEDs on `PB_13`, `PB_14`, and `PB_15` may mirror inputs or the move command. The current REDTAIL mapping is authoritative over older notes.

Assignment

1. Derive a pure policy

Write a truth table for rain and both endpoint sensors. Separate the decision from GPIO access so it can be tested without hardware. Your policy must return one Boolean move request.

2. Create the Mbed application

Create `main.cpp`:

```
#include "mbed.h"

bool should_move(bool rain, bool right_endpoint, bool left_endpoint) {
    // Return the required one-bit policy.
}

int main() {
    // Configure five DigitalIn objects and one DigitalOut.
    // Initialize move low before entering the loop.

    while (true) {
        // Read one stable snapshot of all inputs.
        // Evaluate should_move and update the output.
        ThisThread::sleep_for(10ms);
    }
}
```

Do not repeatedly read a changing input while computing one output value. Initialize `move` low so startup cannot produce unintended motion.

3. Test the policy before programming

Exercise the normal and dual-endpoint rows independently of the GPIO loop. Confirm that changing M or P cannot alter the core result.

4. Compile and program

1. Select `NUCLEO_WB55RG`, the Mbed environment, and the Wiper simulation.
2. Compile and resolve every error or pin-name warning.
3. Program the board and keep the simulation visible.
4. Use diagnostic LEDs if needed to separate an input-mapping problem from a policy problem.

Required test sequence

Test	Action	Expected behavior
Dry start	Start with rain disabled.	Move remains low.
Rain start	Enable rain.	The blade starts moving.
Automatic reversal	Keep rain enabled through both endpoints.	The simulation reverses the blade automatically.
Pause and resume	Disable rain between endpoints, then re-enable it.	The blade pauses and resumes.
Button isolation	Press M and P independently while dry.	The core application remains stopped.
Sensor fault	Trigger both endpoint sensors together.	Move becomes low until the fault clears.
Recovery	Clear the fault while rain remains enabled.	Movement resumes.

Deliverables

Submit the truth table, `main.cpp`, policy-test evidence, live test evidence, and a short explanation of the controller/simulation ownership boundary.

Optional extensions

- Assign M or P a clearly documented manual-wipe behavior.
- Latch a fault indicator until an explicit acknowledgement.
- Report state transitions over serial without slowing the polling loop.
- Replace the delay with a periodic event while preserving one input snapshot.

Completion checklist

- The target and all six pins match the current REDTAIL mapping.
- Move initializes low and every loop uses one input snapshot.
- Rain starts wiping and dry conditions stop it.
- The simulation owns automatic reversal.
- Dual endpoints stop safely and M/P remain optional.
- All source and evidence are ready to submit.